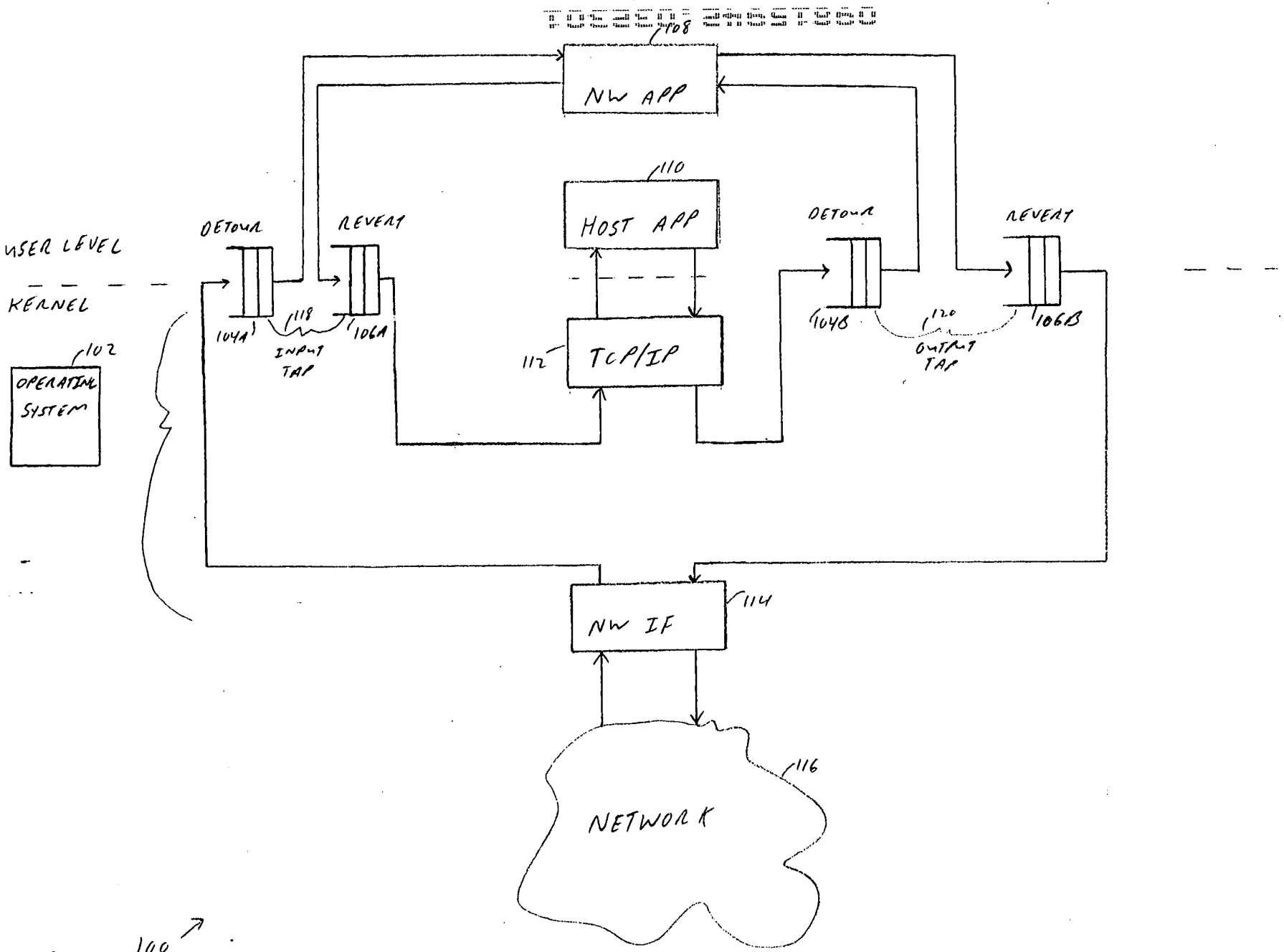

108
NW APP
110
HOST APP
DETOUR
REVERT
USER LEVEL
KERNEL
104A
118
INPUT TAP
106A
112
TCP/IP
104B
120
OUTPUT TAP
106B
102
OPERATING SYSTEM
114
NW IF
116
NETWORK
100

FIG. 1

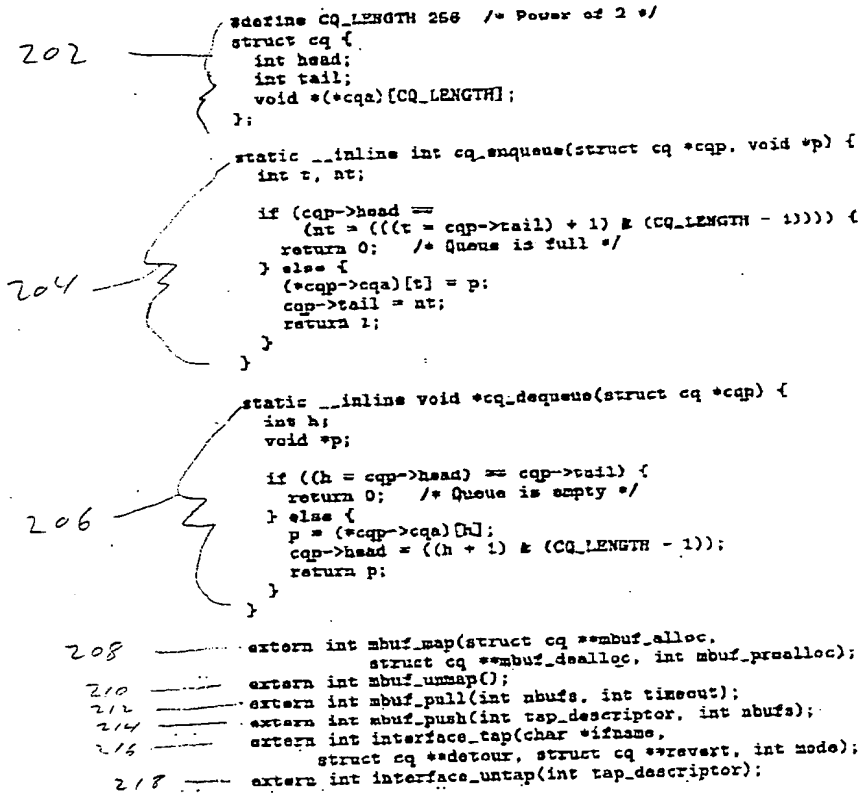
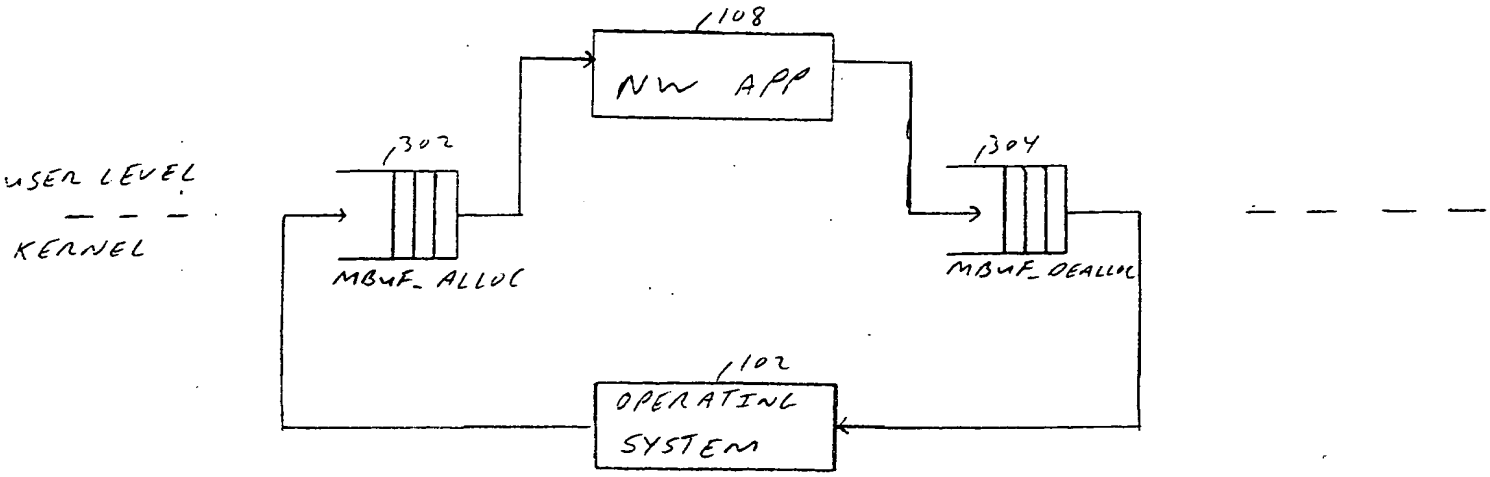
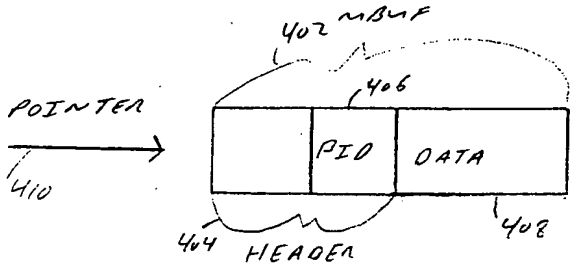

202
#define CQ_LENGTH 256 /* Power of 2 */
struct cq {
int head;
int tail;
void *(*cqa)[CQ_LENGTH];
};
204
static __inline int cq_enqueue(struct cq *cqp, void *p) {
int t, nt;
if (cqp->head ==
(nt = (((t = cqp->tail) + 1) & (CQ_LENGTH - 1)))) {
return 0; /* Queue is full */
} else {
(*cqp->cqa)[t] = p;
cqp->tail = nt;
return 1;
}
}
206
static __inline void *cq_dequeue(struct cq *cqp) {
int h;
void *p;
if ((h = cqp->head) == cqp->tail) {
return 0; /* Queue is empty */
} else {
p = (*cqp->cqa)[h];
cqp->head = ((h + 1) & (CQ_LENGTH - 1));
return p;
}
}
208
extern int mbuf_map(struct cq **mbuf_alloc,
struct cq **mbuf_dealloc, int mbuf_prealloc);
210
extern int mbuf_unmap();
212
extern int mbuf_pull(int nbufs, int timeout);
214
extern int mbuf_push(int tap_descriptor, int nbufs);
216
extern int interface_tap(char *ifname,
struct cq **detour, struct cq **revert, int mode);
218
extern int interface_untap(int tap_descriptor);
200

FIG 2



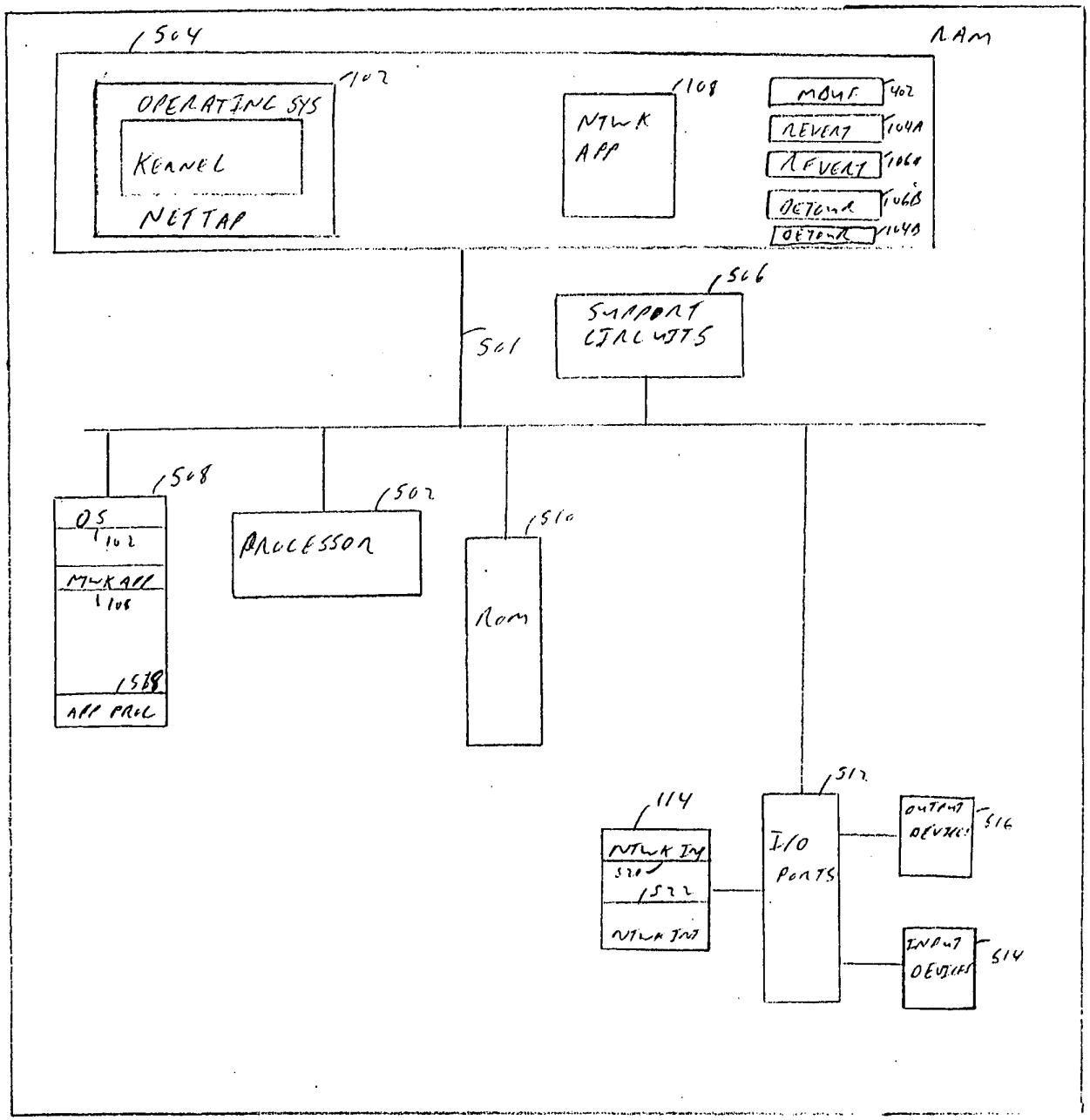
300

FIG. 3



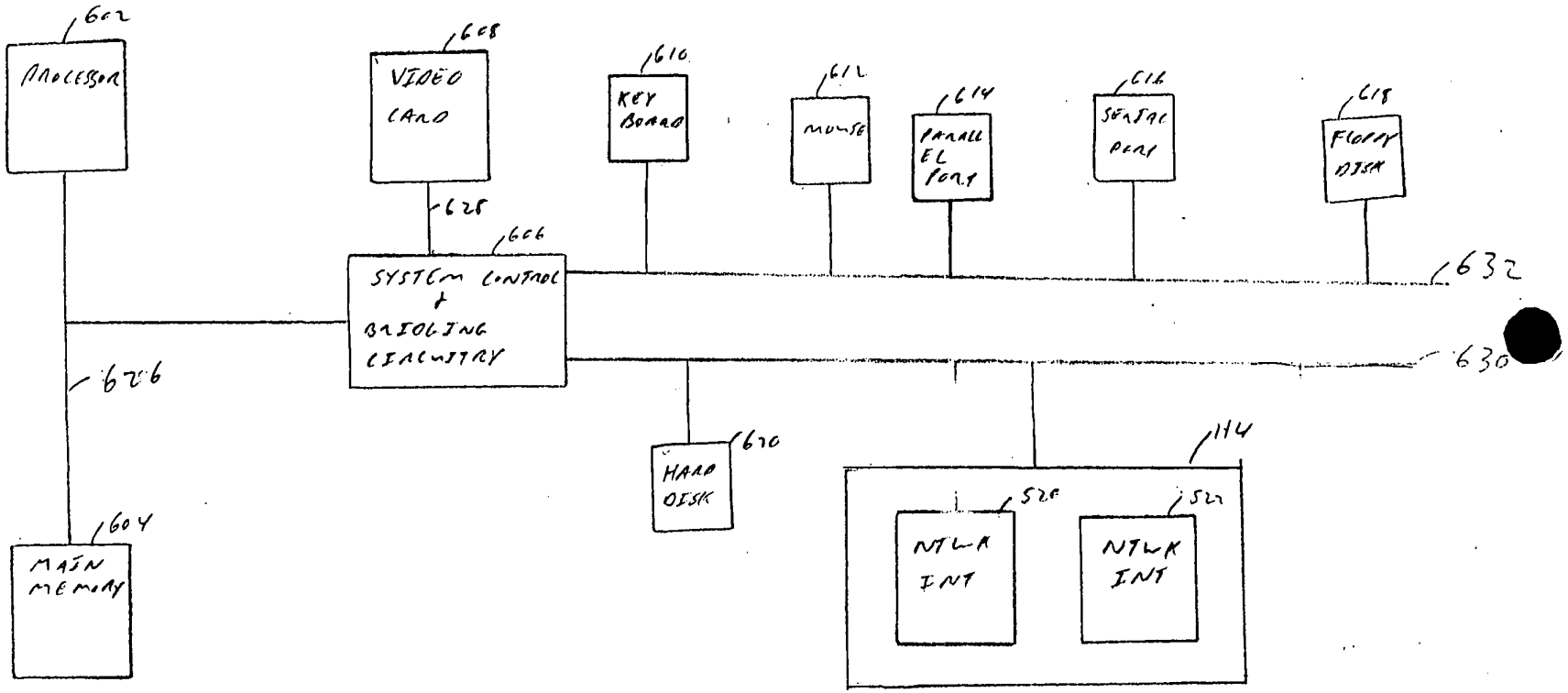
400

FIG. 4


504
RAM
OPERATING SYS
KERNEL
NETTAP
102
NTWK APP
108
MOUT
402
REVENT
104A
REVENT
106A
DETONR
106B
DETONR
104B
506
SUPPORT CIRCUITS
501
508
OS
102
MNKAPP
108
588
APP PROC
502
PROCESSOR
510
ROM
512
114
NTWK IN
520
522
NTWK INT
I/O PORTS
OUTPUT DEVICE
516
INPUT DEVICE
514

500

FIG. 5


602
PROCESSOR
608
VIDEO CARD
610
KEY BOARD
612
MOUSE
614
PARALLEL PORT
616
SERIAL PORT
618
FLOPPY DISK
628
606
SYSTEM CONTROL & BUFFERING CIRCUITRY
632
630
626
670
HARD DISK
114
526
NTWK INT
524
NTWK INT
604
MAIN MEMORY

600

FIG. 6

```
#define PERROR(msg) {perror(msg); exit(1);}

#define BUSY_WAIT_LIMIT 10

#define CLASS_DENY 0
#define CLASS_ACCEPT 1
#define CLASS_HOST -1

int network_application(struct mbuf *m) {
  ... process packet and return its classification ...
}

void drop(struct cq *dealloc, struct mbuf *m) {
  if (cq_enqueue(dealloc, m) == 0) {
    mbuf_push(-1, 0);
    if (cq_enqueue(dealloc, m) == 0) {
      PERROR("Deallocating");
    }
  }
}

main () {
  int i, class, out, nempty, tin[2], tout[2];
  struct cq *alloc, *dealloc, *detour_rx[2],
    *revert_ip_in[2], *detour_ip_out[2], *revert_tx[2];
  char name[] = "fxp0";
  struct mbuf *m;
                                        /* Map mbufs */
  if (mbuf_map(&alloc, &dealloc, 0) != 0) {
    PERROR("Mapping mbufs");
  }
  for (i = 0; i < 2; i++) {             /* Tap interfaces */
    name[3] = (i == 0) ? '0' : '1';
    if ((tin[i] = interface_tap(name, &detour_rx[i],
                        &revert_ip_in[i], TAP_INPUT)) < 0) {
      PERROR("Tapping input");
    }
    if ((tout[i] = interface_tap(name, &detour_ip_out[i],
                        &revert_tx[i], TAP_OUTPUT)) < 0) {
      PERROR("Tapping output");
    }
  }
  for (nempty = 0, i = 0; ; nempty++, i = (i + 1) & 1) {
    if ((m = cq_dequeue(detour_rx[i])) != 0) {/* Rx pkt */
      nempty = 0;
      if ((class = network_application(m)) ==
          CLASS_ACCEPT) {                   /* Bridge pkt */
        out = cq_enqueue(revert_tx[(i + 1) & 1], m);
      } else if (class == CLASS_HOST) {     /* Host input */
        out = cq_enqueue(revert_ip_in[i], m);
      } else {
        out = 0;
      }
      if (!out) {   /* Drop if queue full or pkt denied */
        drop(dealloc, m);
      }
    }
    if ((m = cq_dequeue(detour_ip_out[i])) != 0) {
      nempty = 0;                           /* Host output */
      if (cq_enqueue(revert_tx[i],m) == 0) {  /* Tx pkt */
        drop(dealloc, m);                   /* Drop if queue full */
      }
    }
    if (nempty == BUSY_WAIT_LIMIT) {
      mbuf_pull(0, 0);                      /* Wait for pkt */
      nempty = 0;
    }
  }
}
```

700

FIG. 7